Mobiilipeli Android-käyttöjärjestelmälle

Jussi Ukkola

Kaupan ja kulttuurin toimialan opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
Tradenomi

TORNIO 2013

TIIVISTELMÄ

KEMI-TORNION AMMATTIKORKEAKOULU, Tietojenkäsittely

Koulutusohjelma:	Tietojenkäsittely
Opinnäytetyön tekijä(t):	Jussi Ukkola
Opinnäytetyön nimi:	Mobiilipeli Android-käyttöjärjestelmälle
Sivuja (joista liitesivuja):	36
Päiväys:	29.11.2013
Opinnäytetyön ohjaaja(t):	Yrjö Koskenniemi

Opinnäytetyöni aihe on mobiilipeli Android-käyttöjärjestelmälle. Opinnäytetyöni tavoitteena on, että oppisin mobiilipelin tekemisen vaiheet ja ohjelmoinnin. Työssäni toteutan mobiilipelin teon Android-käyttöjärjestelmälle. Mobiilipelin ohjelmointi on aiheenani, koska olen erittäin kiinnostunut pelien ohjelmoinnista ja peliohjelmoijille on iso kysyntä työmarkkinoilla.

Työssäni tutkin, mitä Java-ohjelmointikielellä on mahdollista toteuttaa mobiilipeli Android-käyttöjärjestelmälle ja mitkä ovat mobiilipelin teossa olevat eri toteutusvaiheet.

Aineistona työssäni käytin useita eri Internet-lähteitä. Keräsin aineistoja etsiessäni Internetistä koodia, jolla toteuttaa mobiilipelin ohjelmointi. Tutkimusmetodina käytin konstruktiivista tutkimusmetodia.

Opinnäytetyötä tehdessäni opin järjestelmällisen toteutuksen ja suunnittelun, mitkä liittyvät ohjelmointiin. Opin myös Java-ohjelmointikielen rajat ja mahdollisuudet peliä ohjelmoidessani.

Asiasanat: Mobiili, ohjelmointi, Java

ABSTRACT

KEMI-TORNIO UNIVERSITY OF APPLIED SCIENCES, Business and Culture

Degree programme:	Bachelor of Business Administration
Author(s):	Jussi Ukkola
Thesis title:	Mobile game for Android OS
Pages (of which appendixes):	36
Date:	29 November, 2013
Thesis instructor(s):	Yrjö Koskenniemi

The topic of my thesis is mobile game for Android OS. The objective of my thesis is to learn the phases of mobile game programming. For completing my thesis, I programmed a mobile game for Android OS. The reason for programming a mobile game is that I am interested in programming games and there is a huge market for mobile programmers in Finland's labor market.

In my thesis, I examine the use of Java coding language in mobile game programming for Android OS and what the phases in mobile game programming are.

As the material in my thesis I used many relevant Internet sources. I collected material while I was searching a suitable code, for programming mobile game, from the Internet. As a research method for my thesis I used constructive research method.

When doing the mobile programming, I learned the systematic implementation and design which are relevant to programming. I also learned the possibilities and limits for using Java as a coding language for my mobile game.

Keywords: Mobile, programming, Java

SISÄLLYS

1 JOHDANTO

Älypuhelinten määrä on kasvanut, ja yhä useammalla on jokin mobiilipeli asennettuna omaan puhelimeen. Ericsson ennustaa, että mobiililiittymien määrä kasvaa vuoteen 2019 mennessä 9,3 miljardiin. Niistä 5,6 miljardia eli yli 60 prosenttia on älypuhelinliittymiä. (Leskinen 2013, hakupäivä 12.11.2013.) Tästä johtuen pelialalle on tullut kysyntää ja alalla onkin työvoimapula tällä hetkellä. Ammattimaisen työvoiman tarve on peliyrityksille tehdyn kyselyn perusteella seuraavan vuoden aikana noin 150 työntekijää (Lehmusvirta 2013, hakupäivä 25.10.2013).

Mielenkiintoni ohjelmointia ja pelejä kohtaan ovat myös syynä, miksi päädyin tekemään mobiilipeliä opinnäytetyönä. Pelejä olen pelannut jo, kun olin pieni poika; aluksi Nintendo Entertainment System:llä (NES), joka on Nintendon kehittämä videopelikonsoli, Marioa ja Megamania pelaten. Ajan kuluessa ja pelikonsoleiden kehittyessä, myös oma pelaaminenkin laajeni ja nykyisin tuleekin pelattua vapaa-ajalla laajasti erilaisia ja erityylisiä pelejä. Ohjelmointia tein ensimäisen kerran peruskoulussa, jossa tehtiin aluksi aivan yksinkertainen HTML-sivu, jonka tarkoituksena oli tutustua ohjelmointiin. Peruskoulun jälkeen jatkoin HTML-ohjelmointia vapaa-ajalla vähän, mutta lukion aikana päätin, että ryhdyn ohjelmoijaksi ja haen johonkin, missä sitä opetetaan. Kemi-Tornion ammattikorkeakoulussa tutuksi tuli PHP- ja Javascript-ohjelmointi. Ohjelmointi oli kiinnostavaa ja harjoittelujakson aikana päätin, että teen opinnäytetyönä mobiilipelin.

Tulen käsittelemään tässä opinnäytetyössä aluksi Javasta: Mikä se on, mitä sillä voi tehdä ja mitkä sen pääpiirteet ovat. Jatkan vielä Javan käsittelyä Android-alustalle ja mikä on ero Android-sovelluksen ja tavallisen Java-sovelluksen ohjelmoinnissa. Java-osion jälkeen kerron ohjelmistotuotannon vaiheista tämän mobiilipelin teossa eli käyn läpi minkä sovelluskehityksen valitsin, vaatimusmäärittelyn teon, riskianalyysin tekemisen ja ohjelmointikielen valinnan perustelun sekä testausmenetelmästä. Tämän jälkeen käyn läpi tekemäni mobiilipelin pääpiirteitä läpi, jossa esittelen pelin koodin toimivuutta.

Pohdin lopuksi luotua peliäni hieman läpi, minkälainen siitä tuli ja vastasiko se asetettuja määrityksiä. Minkälainen prosessi mobiilipelin tekeminen yksin ja vieläpä opinnäytetyönä oli. Lisäksi käsittelen omaa ohjelmoinnin kiinnostusta, inspiroiko tämän

pelin tekeminen ja mitkä mahdollisuudet itsellä on tulevaisuudessa tällä alalla. Pohdinnan lopuksi totean onko tästä opinnäytetyöstä mahdollisesti jollekin apua mobiilipelin teossa.

1.1 Tutkimus- ja tiedonkeruumenetelmä

Käytin opinnäytetyössäni tutkimusmenetelmänä konstruktiivista tutkimusta ja tiedonkeruumenetelmänä Internet-lähteitä. Internet-lähteet koostuivat koodien hakemisesta ja pelialaan liittyvistä artikkeleista. Konstruktiivisuus ilmeni haettujen koodien muokkaamisena peliin sopiviksi.

2 JAVA MOBIILIPELIN OHJELMOINTIKIELENÄ

2.1 Yleistä Javasta

Java on Sun Microsystemsin kehittelemä ohjelmointikieli, joka julkaistiin ensimmäisen kerran vuonna 1995. Ensimmäinen versio, JDK 1.0, soveltui www-sivuille luotavien sovelluksien tekemiseen. Mobiiliohjelmointia varten Javassa on oma ajoympäristö; Java Micro Edition (Java ME). (About Java ME 2013, hakupäivä 20.10.2013.)

Java ME koostuu kolmesta elementistä: asetukset, jotka tarjoavat perus luokkakirjastot ja laajan laitevalikoiman virtuaalikoneelle, ohjelmointirajapintaprofiilit, jotka tukevat kapeahkoa laitevalikoimaa, ja vaihtoehtoiset lisäosat erityisteknologisille ohjelmointirajapinnoille. Ajan myötä Java ME on jaettu kahteen perus asetukseen, pienille mobiililaitteille, joilla on rajoitetut resurssit (laitteen tehokkuus), ja toinen kehittyneemmille mobiililaitteille, kuten älypuhelimille. Pienten laitteiden asetuksia kutsutaan Connected Limited Device Configuration (CLDC) ja kehittyneempien laitteiden asetuksia Connected Device Configuration (CDC). (About Java ME 2013, hakupäivä 20.10.2013.)

2.2 Javalla ohjelmointi

Javalla ohjelmointi koostuu luokista ja niiden olioiden käsittelystä ja kommunikoinnista keskenään. Jotta ohjelmoija pääsee ohjelmoinnissa alkuun, on hänen koneella oltava Java Development Kit (JDK), joka mahdollistaa Java-applettien eli sovelluksia tekemisen, sekä sopiva IDE (ohjelmointiympäristö), jolla toteuttaa koodia (Cho 2013, hakupäivä 10.9.2013). Kun sopiva IDE on valittuna ja koneelta löytyy JDK, voi Javalla aloittaa ohjelmoinnin.

Ohjelmoijan tulee ymmärtää Java-ohjelmoinnissa luokkien ja olioiden käyttö, jotta hän onnistuu luomaan toimivan ohjelman. Luokilla tarkoitetaan Javassa mallia, joka määrittää olioiden toiminnat. Luokkaa voisikin täten kuvailla eräänlaiseksi kaavioksi, joka pitää sisällään erilaisia toimintoja ja rakenteita. Olioilla voidaan täten määritellä näitä toimintoja ja rakenteita tarkemmin. (Cho 2013, hakupäivä 10.9.2013.) Tätä varten

ohjelmoijan tulee kuitenkin suunnitella ja miettiä olioiden sekä luokkien kommunikaatiota, ovatko julkisia vai kommunikoivatko ainoastaan oman luokan sisällä.

Java-ohjelmoinnissa käytetään public-sanaa luokkien ja olioiden edessä, kun halutaan tehdä luokka jonka kanssa muut luokat ja oliot voivat kommunikoida ja interaktioida (The Java® Language Specification 2013, hakupäivä 3.11.2013). Esimerkkinä tästä on omassa pelissä oleva LoadingScreen.java-luokka, joka alustaa Assets.java-luokasta update-metodilla tarvittavat graafiset-asetukset peliä varten.

```
@Override
public void update(float deltaTime) {
    Graphics g = game.getGraphics();
    Assets.menu = g.newImage("menu.png", ImageFormat.RGB565);
    Assets.background = g.newImage("background.png", ImageFormat.RGB565);
    Assets.character = g.newImage("character.png", ImageFormat.ARGB4444);
    Assets.character2 = g.newImage("character2.png", ImageFormat.ARGB4444);
    Assets.character3  = g.newImage("character3.png", ImageFormat.ARGB4444);
    Assets.characterJump = g.newImage("jumped.png", ImageFormat.ARGB4444);

    Assets.tiledirt = g.newImage("tiledirt.png", ImageFormat.RGB565);
    Assets.tilegrassTop = g.newImage("tilegrasstop.png", ImageFormat.RGB565);
    Assets.tilegrassBot = g.newImage("tilegrassbot.png", ImageFormat.RGB565);
    Assets.tilegrassLeft = g.newImage("tilegrassleft.png", ImageFormat.RGB565);
    Assets.tilegrassRight = g.newImage("tilegrassright.png", ImageFormat.RGB565);

    Assets.hardpixel = g.newImage("hardpixel.png", ImageFormat.RGB565);
    Assets.pixelair = g.newImage("pixelair.png", ImageFormat.RGB565);
    Assets.pixelground = g.newImage("pixelground.png", ImageFormat.RGB565);
    Assets.pixelleft = g.newImage("pixelleft.png", ImageFormat.RGB565);
    Assets.pixelright = g.newImage("pixelright.png", ImageFormat.RGB565);


    Assets.button = g.newImage("button.jpg", ImageFormat.RGB565);
```

Kuva 1. LoadingScreen.java-luokka com.peksi.sonofpixel-paketissa, jossa alustetaan Assets.java-luokasta graafisia tietoja peliä varten

Private-sanan käyttö on sen sijaan päinvastainen vaikutus ja tarkoitus kuin public:lla. Ainoastaan luokka, jossa private-sana on, voi kommunikoida ja interaktioida sen kanssa. Privatea käytetään, kun halutaan tehdä luokalle jokin ominaisuus tai toiminta, johon mitkään muut luokat ja oliot eivät pääse ja täten sen ominaisuuden saa rajattua vain tälle kyseiselle luokalle. (The Java® Language Specification 2013, hakupäivä 3.11.2013.) Hyvänä esimerkkinä pelistäni com.peksi.sonofpixel-paketista GameScreen.java-luokasta kohta, jossa määritellään mitä tapahtuu, kun pelaaja koskettaa näytössä määriteltyä kohtaa. Tällöin tämä tapahtuma tapahtuu vain ja ainoastaan tässä luokassa eikä esimerkiksi LoadingScreen.java-luokassa, jolloin pelaaja voisi liikuttaa hahmoa jo silloin, kun peli suorittaa tiedostojen lataamista.

```
private void updateRunning(List touchEvents, float deltaTime) {

    // This is identical to the update() method from our Unit 2/3 game.

    // 1. All touch input is handled here:
    int len = touchEvents.size();
    for (int i = 0; i < len; i++) {
        TouchEvent event = (TouchEvent) touchEvents.get(i);
        if (event.type == TouchEvent.TOUCH_DOWN) {

            if (inBounds(event, 0, 285, 65, 65)) {
                pixelson.jump();
                currentSprite = anim.getImage();
            }


            if (event.x > 400) {
                // Move right.
                pixelson.moveRight();
                pixelson.setMovingRight(true);

            }

        }
```

Kuva 2. GameScreen.java-luokasta kohta, jossa määritellään mitä tapahtuu kun pelaaja koskettaa ruudulla tiettyä kohtaa

Ohjelmoijan on kuitenkin mahdollista hakea tai asettaa jokin arvo toiseen luokkaan private-luokasta, kun hän käyttää get- tai set-metodia. Get-metodi palauttaa aina jonkin arvon, kun taas set-metodi ei palauta mitään arvoa vaan se muuttaa määriteltyä muuttujaa. Ohjelmoijan tulee kuitenkin käyttää get- ja set-metodeja vain silloin, kun jokin muuttuja pitäisi saada toiseen luokkaan ilman, että private-luokka pitäisi muuttaa public-luokaksi. (Cho 2013, hakupäivä 10.9.2013.)

Static-sanalla saadaan Javassa sidottua metodi luokkakohtaiseksi, jolloin on olemassa vain yksi ja ainoa olio, jota muokkatessa se muuttuu myös muissakin luokissa joissa kyseistä oliota on käytetty (The Java® Language Specification 2013, hakupäivä 3.11.2013). Eli esimerkiksi pelissä on luotuna public static Music theme, joka luo theme olion jota käytetään pelissä taustamusiikkina. Kun tätä kyseistä theme:ä sitten muokkaa jossain vaiheessa, se vaikuttaa myös sitten automaattisesti pelin taustamusiikkiin. Kyseistä theme:ä on siis vain yksi ja ainoa olemassa, jonka muokkaus vaikuttaa myös muihin luokkiin joissa theme:ä on käytetty.

Void-sanaa käytetään Javassa, kun jokin metodi ei annakaan mitään arvoa tai lukua. Javassa metodeilla täytyy olla jokin arvotyyppi määriteltynä. Yleensä, kun metodi arvon

palauttaa, se on määritelty esimerkiksi int:ksi (integer). Sen sijaan, kun metodi ei palautakkaan arvoa, se tulee olla tällöin määriteltynä void:ksi. (The Java® Language Specification 2013, hakupäivä 3.11.2013.) Esimerkiksi pelissä hahmon liikkumisen arvotyypiksi on määritelty void, koska jos arvotyypiksi olisi määritelty integer, ohjelma tulostaisi kyseistä arvoa kun pelaaja yrittää liikkua.

Javassa muuttujien määrittely on myös otettava huomioon. Muuttujan tyyppi voi olla oliokohtainen, luokkakohtainen, paikallinen tai muodollinen parametri. Oliokohtainen muuttuja koskee vain ja ainoastaan kyseistä oliota, joten siihen tehdyt muutokset eivät vaikuta muihin olioihin tai luokkiin. (The Java® Language Specification 2013, hakupäivä 3.11.2013.) Esimerkiksi pelissä on viisi uniikkia kenttää, joista yhteen tehdään muutoksia. Nämä muutokset vaikuttavat vain tähä yhteen kenttään, muut kentät pysyvät muokkaamattomina. Luokkakohtainen muuttuja vaikuttaa koko luokkan sisältöön, ei erikseen jokaiseen olioon ja kenttä on määritelty määreellä static. Tällöin kaikki luokan oliot jakavat luokkakohtaisen muuttujan. (The Java® Language Specification 2013, hakupäivä 3.11.2013.) Esimerkkinä pelistä hahmon fysiikkasäännöt, jotka vaikuttavat jokaiseen pelin kenttiin ja ovat täten samat.

Paikallinen muuttuja koskee vain kyseistä metodia, mihin se on luotu. Luokassa voi olla useita metodeja, joille jokaiselle on määriteltynä omat muuttujat. Esimerkkinä pelistä hahmo, jolle on määriteltynä oikealle ja vasemmalle liikkuminen. Molemmilla on sama muuttuja speedX, jolla tarkoitetaan hahmon liikkuvuutta vaakasuunnassa, mutta sille on määriteltynä eri arvot kyseisten muuttujien kohdalla. Paikallisella muuttujaa voidaan kutsua luokan sisällä. Tällöin kyseessä on muodollinen parametri. (The Java® Language Specification, hakupäivä 3.11.2013.) Pelistä esimerkkinä GameScreen muuttuja, jota kutsutaan pelin näyttämiseksi ruudulla GameScreen.java-luokassa.

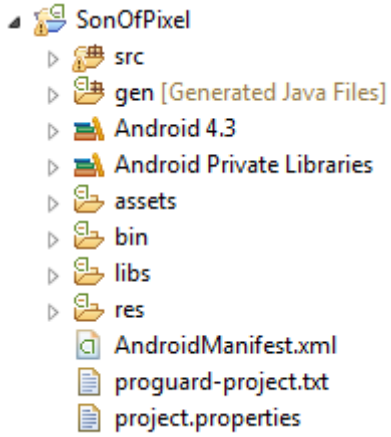
2.3 Android-ohjelmoinnin perusteet lyhyesti

Android on käyttöjärjestelmä puhelimille ja muille mobiililaitteille. Androidin sovelluksissa käytetään Javasta sen luokkakirjastoja muttei virtuaaliympäristöä vaan Androidissa on kehitetty oma jota kutsutaan Dalvik:ksi. Tästä johtuen Android ei tue J2ME-sovellusympäristöä vaan ainoastaan ainut viite Javaan on sen luokkakirjastojen

käyttö. (Android, the world's most popular mobile platform 2013, hakupäivä 1.11.2013.)

Android-sovellusta varten ohjelmoijan pitää hakea itselleen Eclipseen (ks. 2013) lisäosan, joka on Android SDK (Software Development Ki). Android SDK:n mukana tulee tarpeelliset luokkakirjastot, joilla voi toteuttaa erilaisia toimintoja Android-käyttöjärjestelmälle ohjelmoidessa. Tämän lisäksi ohjelmoijalla täytyy olla ADT-lisäosa, joka yhdistää Eclipsen ja Android SDK:n keskenään, tuloksena täysin yhteensopiva ohjelmointiympäristö jossa voi kehitellä Android-käyttöjärjestelmälle sovelluksia. (Android SDK 2013, hakupäivä 1.9.2013.) Tämän jälkeen ohjelmoijan tulee tehdä Eclipsen asetuksiin muutoksia, kuten vaihtaa Javan versio oikeaksi mikäli se ei ole oikea (Cho 2013, hakupäivä 10.9.2013).

Asetuksien ollessa kunnossa, siirrytään tutkimaan luodun projektin rakennetta kuva 3:ssa.



Kuva 3. Projektin rakenne Eclipsessä

Src-kansioon luodaan Java-luokat eli koodin kirjoitus tapahtuu tähä kansioon. Gen-kansio pitää sisällään automaattisesti generoitua koodia, joka helpottaa ohjelmoijan toimintaa merkittävästi Eclipsessä: ohjelmoija kirjoittaa haluttua koodia jonkin verran, jonka jälkeen Eclipse antaa listan vaihtoehdoista, joilla täydentää koodi loppuun. Android 4.3-, Android Private Libraries- ja libs-kansio pitää sisällään Android-luokkakirjastoja. Assets- ja res-kansiot pitävät sisällään resursseja, kuten kuva- ja merkkijono-tiedostoja. Res-kansioon tulee yleensä useammat eri versiot näistä tiedostoista jotta Android-sovellus voisi käyttää niitä esimerkiksi eri kuvaruudun koka varten. Proguard-project.txt-tiedostolla pyritään estämään muiden käyttäjien tekemää

takaisinmallinnusta, jolla pyritään tutkimaan sovelluksen toimintaa ja rakennetta, kuten lähdekoodia. (Cho 2013, hakupäivä 10.9.2013.)

AndroidManifest.xml-tiedosto on projektin runko. Se pitää sisällään paljon informaatiota, kuten projektin versionumeron, SDK-versiovaatimuksen sekä pääaktiviteetin. Pääaktiviteetti on tässä projektissa peli, SampleGame.java-luokka. Pääaktiviteettia on siis jokin suoritettava asia, jonka ympärille koodia sitten rakennetaan, samalainen kuin yksinkertaisessa Java-sovelluksessa jossa päämetodin ympärille rakennetaan sovellus. Android-sovellusta varten tulee tehdä AndroidManifest.xml-tiedostoon käyttöoikeuspyynnöt. Tämä antaa käyttäjälle mahdollisuuden joko hyväksyä tai kieltää nämä pyynnöt. Mikäli käyttäjä ei anna lupaa sovellukselle sen pyytämille käyttöoikeuspyynnöille, ei sovellusta asenneta puhelimeen. Käyttäjän hyväksyessään nämä pyynnöt sovellus asennetaan puhelimelle. (Android, the world’s most popular mobile platform 2013, hakupäivä 1.11.2013.) Ohjelmoijan kannattaakin miettiä tarkkaan, mitä käyttöoikeuksia sovellus tarvitsee sillä niillä voi saada paljon tuhoa aikaiseksi käyttäjän puhelimessa mikäli käyttöoikeuspyynnöt ovat väärät (Cho 2013, hakupäivä 10.9.2013).

2.4 Java ja Android, ohjelmoinnin monipuolisuus

Kun Java- ja Android-ohjelmointi yhdistyvät, saadaan aikaiseksi moniulotteinen ja rakenteinen sovellus, kuten esimerkiksi toteuttamani mobiilipeli. Ohjelmoijalla on käsissään laaja luokkakirjasto, sillä Android-alusta käyttää Javan luokkakirjastoja sovelluksissaan. Lisäksi Eclipse Android SDK Manager-lisäosalla varustettuna luovat todella kattavan ohjelmointiympäristön ohjelmoijalle. Ohjelmoijan haaste onkin kehitellä loogiset ratkaisut luokkien ja olioiden kesken sekä keksiä miten toteuttaa suunnittelemansa koodin.

Mobiilipeli koostuu kahdesta osasta: pelimoottorista, joka koostuu pelin teknisistä toiminnoista, kuten äänentoistosta, ja pelin lähdekoodista, joka nivoo yhteen pelin ja tekee siitä toimivan kokonaisuuden. Pelimoottorin ominaisuudet riippuvat pitkälti siitä, minkälaisen pelin haluaa tehdä ja mitä ominaisuuksia haluaa erityisesti peliä varten olevan. Pelin lähdekoodissa sitten määritellään pelille uniikit ominaisuudet, kuten hahmo ja pelin äänet. Samaa pelimoottoria voidaan käyttää myös toisessa vastaavassa

pelissä. Ohjelmoijan tarvitsee tällöin vaihtaa pelin lähdekoodi-osio eikä tarvitse koskea pelimoottoriin asetuksiin ellei koe tarvetta tehdä hienosäätöä. (Cho 2013, hakupäivä 10.9.2013.)

Luokkien periytymisominaisuudella, jolla tarkoitetaan luokan piirteiden (kenttien ja metodien) siirtymistä toiselle luokalle, voidaan luoda monia aliluokkia, laajennettuja tai johdettuja luokkia yliluokasta. Luokkien periytyminen luodaan käyttäen komentoa extends. (Cho 2013, hakupäivä 10.9.2013.) Tällä on mahdollista luoda luokka, kuten esimerkiksi vihollinen, jolle luodaan aliluokaksi esimerkiksi rosvo. Tällöin rosvo-luokka perii vihollinen-luokasta kaikki sen ominaisuudet (esimerkiksi pelaajalle vihamielinen, jolloin se pyrkii ”tappamaan” pelaajan) ja vielä voidaan määrittää rosvo-luokalle vain sitä koskevia ominaisuuksia (esimerkiksi paljonko rosvolla on elämäpisteitä). Aliluokalle on vielä mahdollista luoda oma aliluokka, esimerkiksi rosvovalkoinen-luokka, joka perii sen rosvo-luokasta sen ominaisuudet (huomioitavaa on, että myös vihollinen-luokasta ominaisuudet periytyvät tällöin rosvovalkoinen-luokalle) ja voidaan määrittää rosvovalkoinen-luokalle omia ominaisuuksia (rosvovalkoisen väri on valkoinen).

Luokkien periytymisominaisuus ei ole ohjelmoijan ainut keino luoda moniulotteisia luokkasuhteita. Import-komennolla on mahdollista tuoda luokkaan muita luokkia sekä olioita. Import on tärkeä myös sen takia, että sillä tuodaan Java- ja Android-luokkakirjastoja, esimerkiksi pelimoottoriin Android-alustan ominaisuuksia. Importin etuna luokan periytymisominaisuuteen on, että sillä voidaan tuoda luokasta pelkästään yksi ominaisuus eikä kaikkia. (Cho 2013, hakupäivä 10.9.2013.) Tästä havainnollistavana esimerkkinä pelissäni com.peksi.framework.implementation-paketin SingleTouchHandler.java-luokka, jossa tuodaan käyttöön Java- ja Android-luokkakirjastosta sekä com.peksi.framework-paketista toimintoja valikoiden.

```
package com.peksi.framework.implementation;

import java.util.ArrayList;
import java.util.List;

import android.view.MotionEvent;
import android.view.View;

import com.peksi.framework.Pool;
import com.peksi.framework.Input.TouchEvent;
import com.peksi.framework.Pool.PoolObjectFactory;
```

Kuva 4. Import-komennolla tuodaan käyttöön SingleTouchHandler.java-luokkaan Java- ja Android-luokkakirjastoista sekä com.peksi.framework-paketista toimintoja valikoiden

Ohjelmoijan luodessa pelimoottoria, tulee hänen käyttää implements-komentoa. Implements-komennolla tuodaan rajapintoja, joiden tuonti ei onnistu extends-komennolla, jota käytetään luokkien tuomiseen ja periytymiseen. Implements-komennolla on myös mahdollista tuoda useampi rajapinta kerralla. Implements- ja import-komentoa käyttäen, ohjelmoijalla on mahdollisuus luoda tehokkaita ratkaisuja pelimoottorin osalta. (Cho 2013, hakupäivä 10.9.2013.)

Ohjelmoijan luodessa luokkasuhteita käyttäen import-, extends- ja/tai implements-komentoa voi tulla tilanne, jolloin aliluokalla on saman niminen metodi kuin yliluokalla ja/tai import- ja/tai implements-komennolla tuotu luokka. Tällöin aliluokassa oleva metodi astuu voimaan eikä yliluokan metodi periydy. Ohjelmoija haluaa kuitenkin tämän aliluokan metodin käyttötarkoituksen olevan sama kuin tuodusta luokasta, joten hänen tulee käyttää komentoa @Override. @Override-komennolla voidaan ylikirjoittaa aliluokan metodin, joka muutoin korvaisi extends-, implements- ja/tai import-komennolla luokasta tuodun metodin, jolloin aliluokan metodi vastaa yliluokan metodia. (Cho 2013, hakupäivä 10.9.2013.) Esimerkkinä peliin tehty komentorivi public void update(float deltaTime), jota tarvitaan monessa luokassa jotta luokka toimisi oikein. Ohjelmoija on tehnyt useita luokkasuhteita käyttäen import- ja extends-komentoa ja hän kohtaa ongelman; monella näillä luokilla on sama public void update(float deltaTime)-komentorivi. Hän kuitenkin haluaa public void update(float deltaTime)-komentorivin olevan aliluokissa sama kuin yläluokassa oleva joten hän kirjoittaa ennen jokaista public void update(float deltaTime)-komentoriviä @Override. Täten hän saa public void update(float deltaTime)-komentorivit koskemaan jokaista aliluokkaa, esimerkiksi LoadinScreen.java- ja GameScreen.java-luokat, joissa kyseinen komentorivi on.

3 PROJEKTIN ALOITUS

3.1 Sovelluskehitysmenetelmän valinta

Pelin suunnitteluun ja toteutukseen valitsin Scrumin, sillä toteutan peliä pala kerrallaan kohti kokonaisuutta. Scrum on osa ketteriä ohjelmistokehityksiä, joissa jaetaan ohjelmistonkehitys lyhyisiin iteraatioihin. Iteraatioita tässä projektissa ovat projektisuunnittelu, vaatimusanalyysi, ohjelmistosuunnittelu, koodaus ja testaus. Tutkimusmenetelmänä käytän konstruktiivista tutkimusta, sillä se on luonteeltaan soveltavaa tutkimusta, jossa päämäärä on tiedossa, mutta se miten siihen päästään, ei ole. Nämä kaksi, Scrum ja konstruktiivinen tutkimus, tukevat pelin tekemistä ja suunnittelua, sillä pelin tekeminen koostuu koodin teosta ja sen työstämisesta haluttuun lopputulokseen, joka tässä tapauksessa on toimivan pelin tekeminen.

3.2 Ohjelmointikielen valinta

Ohjelmointikielen valintaan vaikuttivat oma tuntemus ja kokemus sekä kohde alusta. Alunperin oli tarkoitus käyttää C#-ohjelmointikieltä, mutta koska kyseiselle kielelle ei ollut ilmaista ohjelmaa, joka pystyisi konvertoimaan Androidille kyseistä kieltä, valitsin Javan. Muitakin ohjelmointikieliä olisi ollut tarjolla, kuten Ruby, Lua ja Python, mutta niiden oppiminen olisi kestänyt liian kauan verrattuna Javaan.

3.3 Mobiilipelin teossa käytettävät sovellukset

Mobiilipeliä varten oli etsittävä sopivat sovellukset, joilla pystyisin toteuttamaan pelin. Oli haettava teksti-, ääni- ja kuvaeditorit sekä testausta varten mobiililaite. Tärkeimpänä kriteerinä oli edellä mainittujen sovelluksien ja laitteiden käytettävyys sekä hinta. Opiskelijana tyydyin etsimään avoimen lähdekoodin sovelluksia, joilla luoda peliin koodia, musiikki- ja kuvatiedostot.

Tekstieditoriksi valitsin Notepad++:n (ks. 2013), joka on ilmainen tekstieditori ja joka tukee useita eri ohjelmointikieliä. Vielä ei kuitenkaan ole kehitetty mitään koodin

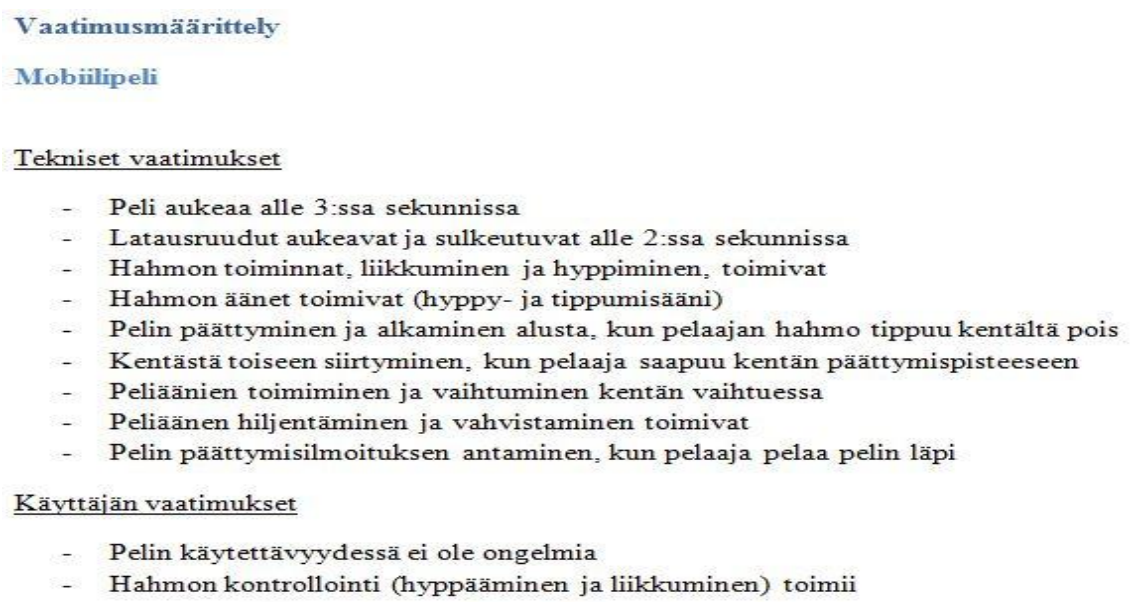
kääntämis- ja sovitusversiota Androidille, joten tulen käyttämään Notepad++:aa lähinnä pelin kenttien .txt-osuuksien tekemiseen. Jotta Javalla ohjelmointi Android-käyttöjärjestelmälle olisi mahdollista, piti sitä varten hakea sopivampi ohjelma kuin Notepad++. Päädyin Eclipseen (ks. 2013), johon hain SDK Manager-lisäosan. Eclipse on, kuten Notepad++, ilmainen ja tukee myös monia ohjelmointikieliä, mutta lisäksi siihen on liitetty debug ja testisimulaattorien käyttö. SDK Manager lisää tähän vielä Androidia varten sopivat projektitiedostot ja luokkakirjastot, joilla on mahdollista luoda Android-käyttöjärjestelmällä toimivaa koodia ja lisäksi mahdollisuus luoda virtuaalisia Android-käyttöjärjestelmäympäristöjä, joissa voisi testata luotua sovellustaan.

Kyseisten virtuaalisten käyttöjärjestelmien käyttö on kuitenkin erittäin hidasta, sillä ohjelmalla kestää erittäin kauan luoda ja avata ne, joten koodin ja pelin testaamista varten, päädyin käyttämään omaa puhelintani, joka on Samsung Galaxy S2 jossa käyttöjärjestelmän versio on 4.1.2. Jotta testaaminen onnistuisi myöhemmin puhelimella, piti puhelimen asetuksista käydä vaihtamassa Sovelluskehittäjien asetukset -kohdasta USB-virheenkorjaus-asetus päälle. Tämä mahdollisti sovelluksen siirtämisen ja ajamisen koneelta puhelimeen USB-johdon avulla.

Kuvien ja äänien tekemiseen en varannut projektissa kovin paljoa aikaa, vain välttämättömien asioiden tekemiseen kuten hahmo, kenttien kuvat ja vähän ääniä. Kuvien tekemiseen käytin Paint-sovellusta (ks. 2013), sillä se oli jo valmiiksi asennettuna koneelle ja sen käyttäminen on hyvin helppoa. Äänien tekemiseen käytin Audacity (ks. 2013) -sovellusta, joka on ilmainen ja olen sitä aiemminkin käyttänyt.

3.4 Vaatimusanalyysi

Vaatimusanalyysissä tein pelille vaatimusmäärittelyn. Vaatimusmäärittelyt tehdään yleensä asiakkaan kanssa, mutta koska tätä opinnäytetyötä ei ole hankkeistettu, tein vaatimusmäärittelyn omien vaatimusten. Mobiilipelissä vaatimukset kohdistuivat lähinnä teknisiin toteutuksiin ja niiden toimivuuksiin. Kuva 5:ssä näkyy pelin vaatimusmäärittely.



Vaatimusmäärittely

Mobiilipeli

Tekniset vaatimukset

- Peli aukeaa alle 3:ssa sekunnissa
- Latausruudut aukeavat ja sulkeutuvat alle 2:ssa sekunnissa
- Hahmon toiminnat, liikkuminen ja hyppiminen, toimivat
- Hahmon äänet toimivat (hyppy- ja tippumisääni)
- Pelin päättyminen ja alkaminen alusta, kun pelaajan hahmo tippuu kentältä pois
- Kentästä toiseen siirtyminen, kun pelaaja saapuu kentän päättymispisteeseen
- Peliäänien toimiminen ja vaihtuminen kentän vaihtuessa
- Peliäänen hiljentäminen ja vahvistaminen toimivat
- Pelin päättymisilmoituksen antaminen, kun pelaaja pelaa pelin läpi

Käyttäjän vaatimukset

- Pelin käytettävyydessä ei ole ongelmia
- Hahmon kontrollointi (hyppääminen ja liikkuminen) toimii

Kuva 5. Mobiilipelilleni laatimani vaatimusmäärittely

Vaatimusmäärittelyn tekeminen auttaa ohjelmistoprojekteissa, sillä se on edullinen tehdä ja yleensä tekninen toiminta tulee muuttumaan projektin edetessä kuten tämänkin pelin teon aikana tuli huomattua. Huolimatta teknisistä muutoksista vaatimusmäärittelyn vaatimukset toteutuivat, sillä ne eivät olleet niin erityisen tarkat ja yksityiskohtaisesti määritelty.

3.5 Ongelmien ennakointi ja niiden ratkaiseminen

Yksi tärkeimmistä asioista ohjelmoinnissa on hyvin tehty suunnittelu ja ongelmien ennakoiminen ja niiden ratkaiseminen. Ongelman ratkaisuun voi huonosti varautuessa tuhlata paljon aikaa sekä hermoja, joten niiden ennakoiminen ja ratkaisujen keksiminen etukäteen edesauttavat pelin ohjelmointia. Taulukko 1:ssä näkyy tekemäni riskianalyysi.

Riskien kartoitus ja analysointi

*) P=Pieni, K=Kohtuullinen, S=Suuri/Huomattava

Riski	Todennäköisyys *)	Seuraus-vaikutus *)	Syyt	Syiden ennaltaehkäisy, suojaustoimenpiteet	Seurauksiin varautuminen, varmistamistoimenpiteet
Projekti ei valmistu ajallaan	K	S	Aikataulussa ei pysytty	Aikataulun luominen ja säännöllinen tarkastelu, missä vaiheessa olen menossa	Säännöllinen tilannekatsaus
En ole tyytyväinen tuotokseeni	K	K/S	Peli ei ole mieleeni	Pidettävä mielessä, että kyseessä vasta ensimmäinen itse tekemä peli	Itsekehu ja maltillisen tavoitteen ylläpitäminen. Tärkeintä on oppiminen!
Tärkeät tiedostot katoavat	P	S	Huolimattomuus	Varmuuskopiointi	Varmuuskopioiden käyttö
Laitteisto hajoaa	K	S	Kone hajoaa	Koneen kunnossa pitäminen	Ennakointi paras tapa, opiskelijabudjetilla ei muuta vaihtoehtoa

Taulukko 1. Opinnäytetyötä varten tehty riskianalyysi

Yksi suurimmista riskeistä, mikä pelin tekemistä koski, oli tiedostojen katoaminen tai tuhoutuminen. Koodin tekeminen ilman apuohjelmia on raskasta ja olisi todella harmillista, mikäli tärkeitä tiedostoja tuhoutuisi ja joutuisi uudestaan kirjoittamaan niitä. Täten otin säännöllisesti varmuuskopioita tiedostoista ulkoiselle kovalevylle sekä toiselle koneelle. Vaikka säännöllinen varmuuskopiointi on työlästä, kun useampaan paikkaan joutuu päivittämään, oli se sen arvoista sillä jouduin kerran turvautumaan varmuuskopioon.

Laitteistojen hajoaminen oli myös yksi merkittävä riski koskien opinnäytetyöni valmistumista, sillä ilman asianmukaista konetta ohjelmointi ei onnistuisi ja en pystyisi peliä ohjelmoimaan. Pyrinkin siis ennakoimaan mahdollisia hajoamisia pitämällä koneen hyvässä kunnossa, sillä opiskelijabudjetilla ei ole varaa mennä ostamaan uusia osia tai kokonaan uutta konetta ja huolto veisi pahimmillaan todella paljon aikaa.

Kun kyseessä on ohjelmointi ja erityisesti pelin ohjelmointi, on omat odotuksetkin luonnollisesti korkealla. Oli haaste pitää oma vaativuus hillityllä tasolla, sillä liiallisten odotusten ja itseltä vaatiminen oli myös opinnäytetyön edistymisen kannalta riskinä. Odotukset peliä kohtaan eivät saaneet missään tapauksessa olla ”Tästä tulee uusi Angry Birds” vaan ennemminkin ”Otan tämän oppimisen kannalta ja teen omien kykyjen hieman ylittävän pelin”. Lisäksi oli hankalien ongelmien ilmetessä pidettävä itsekuria eikä annettava turhautumisen vaikuttaa liikaa opinnäytetyön etenemiseen.

3.6 Testaamisen toteuttaminen

Testaamisessa pyritään löytämään sovelluksesta mahdollisia virheitä ja virheetön testaus tarkoittaa yleensä testin olevan huonosti toteutettu. Testaamisessa käy myös ilmi, mikäli jokin vaatimus ei sovelluksessa toteudukaan vaatimusmäärittelyn mukaisesti. Testaustapoja on laatikkomalli, integraatiotestaus, regressiotestaus, hyväksymistestaus, toimintaan liittymätön testaus ja destruktiivinen testaus.

Testausta varten tehdään myös testisuunnitelmat, joita noudattaen edetään testauksessa. Tätä projektia varten tein kuitenkin vain hyvin yksinkertaisen suunnitelman: Testataan sitä mukaan kun palanen valmistuu. Mahdolliset virheet korjataan ja testataan uudestaan, kunnes virheitä ei enää ole. Lopuksi, kun peli on valmis, suoritetaan vielä lopputestaus, jossa korjataan vielä viimeiset virheet joita voi ilmetä yhteensopivuusongelmina ja muina pieninä huolimattomuuksina, kuten pisteiden puuttuminen tai väärässä kohtaa oleva merkki.

Toteutin testaamisen käyttäen Samsung Galaxy S2:ta testilaitteena ja Eclipseä. Puhelimella kokeilin, että pelin osat toimivat puhelimessa ja Eclipsestä tarkkailin consolen kautta mahdollisia virheilmoituksia joita ei puhelimella näe. Kun testattava osa oli testattu, korjasin mahdolliset virheet ja jatkoin seuraavaan kohtaan ohjelmoinnissa. Lopputestaus tapahtui samalla tavalla, mutta oli mittakaavassaan suurempi, joskin virheiden määrä oli vähäinen ja täten testaukseen ei mennyt ylimääräistä aikaa.

4 MOBIILIPELIN TEKEMINEN

4.1 Peliteeman valinta

Projektisuunnittelun jälkeen oli aika keskittyä itse pelin tekemiseen. Ensimmäiseksi minun piti valita pelille jonkilainen teema, minkä mukaan tehdä pelin hahmo, äänet ja kentät. Teeman valintaan vaikutti vahvasti käytettävissä olevat ohjelmat ja omat kyvyt. Tästä johtuen päädyin melko yksinkertaiseen tyyliin, sillä en ole liiemmin tehnyt mitään grafiikoita tai ääniä aiemmin eivätkä niiden prioriteetti ollut korkealla. Peliteemana on täten pikselit.

Peliteeman vaihtoehtoina oli myös muitakin, mutta pelin kannalta tärkeintä oli löytää teema, jonka oikeasti pystyisin toteuttamaan enkä vain yrittää tehdä jotain mikä olisi kyllä hieno idea. Eikä yksinkertainen teema ole mobiilipelille esteenä, se voi itseasiassa olla sen vahvuuskin. Esimerkkinä Hill Climbing Racing -peli, jonka oululainen peliyhtiö Fingersoft on kehittänyt, on ladattu maailmanlaajuisesti 100 miljoonaa kertaa (Naalisvaara 2013, hakupäivä 28.10.2013).

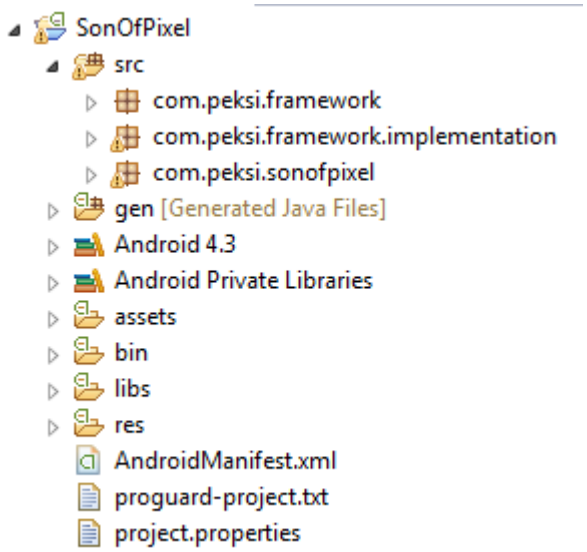
4.2 Pelin juoni

Teeman valittuani oli seuraavaksi määriteltävä pelin idea. Mitä haluaisin, että pelissä tehtäisiin ja miksi. Olen pelannut useampaa eri peliä joten halusin ettei pelini olisi liian samanlainen kuin muut sillä pelialalla on myös erotuttava muista. Oli kuitenkin huomioitava, ettei peli olisi aivan liian erikoinen vaan mahdollisimman helppo lähestyttävä. Täten päädyin tasoloikkapeliin, jossa tavoitteena on päästä kenttä läpi. Pelkästään tämä olisi kuitenkin ollut liian suppea peliksi, joten pelissä pitää olla myös muutakin kuin pelkkä kentän läpäiseminen. Siksi kentissä on paljon erilaisia bonus-objekteja, joita pelaaja voi tavoitella.

Pelin idea vaikutti vahvasti siihen, kuinka hankala pelistä tuli pelaajille. Pelin voi pelata vaivatta läpi, mutta on mahdollista myös bonusobjekteja toteuttaen vaikeuttaa peliä omaa tasoa vastaavaksi. Pääsyynä tähän on oma kokemus pelatuista peleistä, joista sitten halusin yhdistää tähän peliin sopivasti kyseisiä asioita.

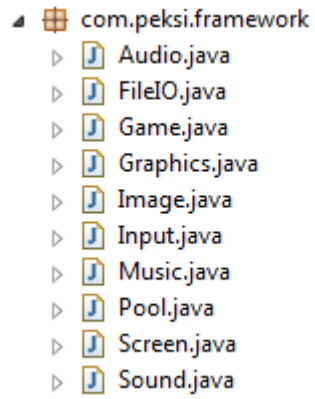
4.3 Ohjelmoinnin aloitus

Peliteeman ja juonen valinnan jälkeen oli seuraavaksi aika aloittaa ohjelmointi. Aluksi lähdin liikkeelle pelimoottorin tekemisestä, sillä ilman sitä peli ei toimi ollenkaan. Pelimoottorin tekeminen alusta asti itse olisi kuitenkin viennyt todella paljon aikaa, joten etsin peliä varten sopivaa pelimoottoria Internetistä. Peliin sopiva pelimoottori löytyikin www.kilobolt.com-sivulta ja samalla myös hyvin selittävä tutoriaali pelin tekemisestä Android-käyttöjärjestelmälle. Tutoriaalia käydessä läpi opin erityisen paljon pelimoottorien toiminnasta ja sellaisen tekemisestä. Kuva 6:ssa näkyy src-kansio, joka pitää sisällään pelin asetukset ja toiminnat.



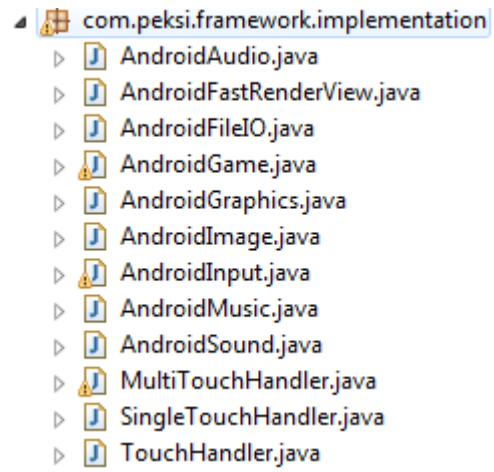
Kuva 6. Pelin src-kansio, joka pitää sisällään pelin toiminnat ja asetukset

Com.peksi.framework-paketti pitää sisällään pelin toiminnan kannalta välttämättömimmät asiat, joita ei kuitenkaan ole vielä tarkasti määritelty mitä ne tulevat tekemään, ovat siis eräänlaiset ääriviivat ja rajat asioille. Game.java-luokassa on määritelty, mitä käyttöliittymään liittyviä luokkia siihen liittyy. FileIO.java-luokan ansiosta try- ja catch-metodeja ei tarvita ja SharedPreferences-luokan, joka on Android-luokkakirjastosta, avulla voidaan käsitellä preferenssi-dataa. Kuva 7:ssä näkyy com.peksi.framework-paketin sisältö.



Kuva 7. Com.peksi.framework-paketin sisältö, joka pitää sisällään niin sanotusti teknisen rungon

Com.peksi.framework.implementation-paketissa tuodaan com.peksi.framework-paketissa tehdyt luokat Android-alustan käyttöön. Android –alkuisissa luokissa, kuten AndroidAudio.java, luodaan yhteensopivuus com.peksi.framework-paketin vastaavan luokan kanssa. Eli esimerkiksi AndroidAudio.java-luokka hakee Audio.java-luokasta asetuksia käytettäväksi Android-alustalle.



Kuva 8. Com.peksi.framework.implementation-paketti, jossa luokat tuovat com.peksi.framework-paketin luokkia käyttöön ja luovat täten yhteensopivuuden Android-alustalle

AndroidGame.java-luokka on yksi tärkeimmistä luokista, mitä tämä mobiilipeli sisältää, ja se on aliluokka Activity-luokasta. Activity-luokka on olenainen osa Android-sovelluksia sillä sen avulla luodaan sovelluksen interaktiivinen-ikkuna, jota ilman käyttäjä ei voisi sovellusta nähdä ja käyttää omalla puhelimella. AndroidGame.java-luokassa määritetään muun muassa näytön resoluutio ja onResume()- ja onPause()-metodit, jotka ovat osa Android-sovellusta. OnPause()-metodissa määritellään pelin

pysäyttäminen, jonka avulla käyttäjä voi laittaa pelin pauselle. OnResume()-metodi kumoaa onPause()-metodin ja käyttäjä voi taas jatkaa pelin pelaamista.

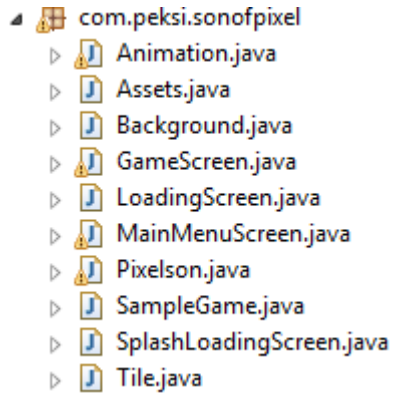
```
import android.app.Activity;

public abstract class AndroidGame extends Activity implements Game {
    AndroidFastRenderView renderView;
    Graphics graphics;
    Audio audio;
    Input input;
    FileIO fileIO;
    Screen screen;
    WakeLock wakeLock;
```

Kuva 9. AndroidGame.java-luokasta kohta, jossa määritellään AndroidGame aliluokaksi Activity-luokasta ja implements-metodilla tuodaan Game.java-luokka

AndroidGraphics.java-luokan avulla luodaan pelin grafiikkaa ja määritellään eri kuvaformaatit kuville. Tällä mahdollistetaan kuvien käyttämät muistin hallinta sovelluksessa, sillä vaikka puhelimessa olisi monta gigaa muistiaa, vain osa siitä on sovelluksen käytössä. Aiemmin com.peksi.framework-paketissa luodussa Graphics.java-luokassa on luotuna kolme kuvaformaattia: ARGB4444, ARGB8888 ja RBG565, ja tämä luokka tuodaan AndroidGraphics.java-luokan käyttöön implements-metodilla. Formaattien numerot tarkoittavat kuvan ulottuvuuksia, leveys, pituus ja syvyys. Ohjelmoijan tuleekin huomioida näiden myötä, paljonko kuvan pixelit vievät näiden myötä. RBG565-formaatti käyttää vähiten muistia ja kuvien laatu on tällöin kohtuu karkea. ARGB4444- ja ARGB8888-formaatit vievät jo enemmän muistia sillä kuvien laadut ovat näissä huomattavasti paremmat kuin RBG565-formaatissa. ARGB8888-formaatin käyttöä tulisi käyttää harkitusti, sillä se vie paljon muistia. Esimerkiksi kuva, jonka koko on 1000 kertaa 1000 pixeliä ja sille on määritelty ARGB8888-formaatti, käyttää tällöin 4MB. Ja mikäli näitä olisi neljä ja laitteen muistia olisi käytettävissä vain 16MB, tapahtuu sovelluksen kaatuminen. (Cho 2013, hakupäivä 10.9.2013.)

Com.peksi.sonofpixel-paketti sisältää luokat, jotka tuovat com.peksi.framework- ja com.peksi.framework.impelementation-paketeista luokkien asetuksia käyttöön. Näiden kaikkien yhdistyessä saadaan luotua toimiva peli Android-alustalla. Sonofpixel-paketin luokat määrittelevät pelin ominaisuudet, kuten hahmon ja äänet, ja on täten vain kyseistä peliä varten. SampleGame.java-luokka on pelin pääaktiviteetti, joka on aliluokka AndroidGame.java-luokasta. SampleGame.java-luokka suoritetaan täten, kun pelin käynnistää. Assets.java-luokka hallinnoi käytettäviä resursseja, mitä pelissä on,

kuten kuvat ja äänet. Nämä alustetaan LoadingScreen.java-luokassa, kun peli käynnistetään. MainMenuScreen.java-luokka pitää pelin alkuvalikon tiedot sisällään, jolloin tähän lisäämällä esimerkiksi info-osan, jossa kerrottaisiin pelin pelaamisen perusteista, voitaisiin toteuttaa lisäämälle sinne vaadittavia koodin osia. GameScreen.java-luokalla luodaan pelin ruudulle näytettäviä asioita ja tapahtumia, kuten ruutu, jossa kerrotaan pelaajan epäonnistuneen (hahmo tippui reiästä alas, jolloin hahmo kuoli). SplashLoadingScreen.java-luokka on aliluokka Screen.java-luokasta ja sen tarkoituksena on alustaa LoadingScreen.java-luokka. LoadingScreen.java-luokassa tapahtuu muut pelin asioiden alustukset pelin käyttöön, kuten hahmon ulkoasu ja äänet. Kuva 10:ssä näkyy com.peksi.sonofpixel-paketin sisältö.

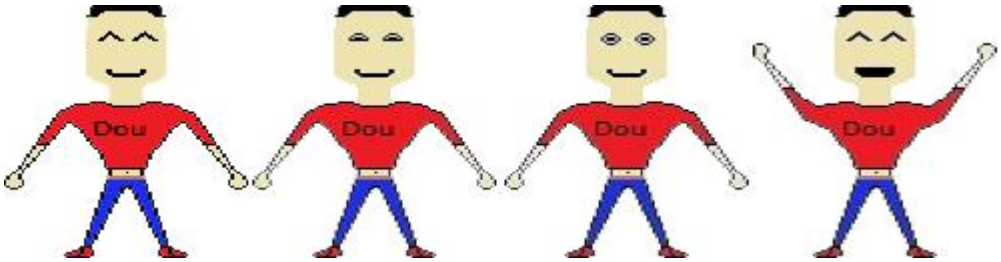


Kuva 10. Com.peksi.sonofpixel-paketti, joka pitää sisällään Son of Pixel-pelin asioita, kuten hahmon ja kentän

Pelimoottoria voi käyttää muihinkin peleihin, jolloin ainoastaan com.peksi.sonofpixel paketti pitää poistaa ja tehdä uusi tilalle. Tällöin voidaan tehdä aivan erilainen peli niin ulkonäöllisesti kuin toiminnallisesti.

4.4 Hahmon luominen

Hahmon luomisessa piti ottaa huomioon koko, muoto ja mahdolliset animaatiot hahmon liikkeissä. Hahmosta tuli lopulta hyvin yksinkertainen, johtuen jo edellä mainituista omista kyvyistä ja käytettävissä olevista sovelluksista, mutta kuitenkin aika persoonallinen. Hahmolle ei lopulta kertynyt paljoa animaatiota, sillä koin etten kerkeäisi tehdä niitä huolellisesti ja saisi niitä kaikkia toimimaan.



Kuva 11. Hahmon ulkonäkö

Hahmo pystyy liikkumaan oikealle ja hyppäämään. Vasemmalle päin liikkuminen ei onnistu, sillä peli on suunniteltu siten että hahmon on liikuttava eteenpäin. Hahmolla on hyppyanimaatio sekä pieni animaatio kasvoissa, jossa hahmon silmät aukeavat ja menevät kiinni. Kuva 12:ssa näkyy hahmon liikettä koskevaa koodia, joka tapahtuu public void update-komennon sisällä. Kuva 13:ssa taas tapahtuu hahmon animaation määrittely.

```
public void update() {
    // Moves Character or Scrolls Background accordingly.

    if (speedX < 0) {
        centerX += speedX;
    }
    if (speedX == 0 || speedX < 0) {
        bg1.setSpeedX(0);
        bg2.setSpeedX(0);

    }
    if (centerX <= 200 && speedX > 0) {
        centerX += speedX;
    }
    if (speedX > 0 && centerX > 200) {
        bg1.setSpeedX(-MOVESPEED / 5);
        bg2.setSpeedX(-MOVESPEED / 5);
    }

    // Updates Y Position
    centerY += speedY;

    // Handles Jumping

        speedY += 1;

    if (speedY > 3){
        jumped = true;
    }

    // Prevents going beyond X coordinate of 0
    if (centerX + speedX <= 60) {
        centerX = 61;
    }
```

```
public void moveRight() {
        speedX = MOVESPEED;
}

public void moveLeft() {
        speedX = -MOVESPEED;
}

public void stopRight() {
    setMovingRight(false);
    stop();
}

public void stopLeft() {
    setMovingLeft(false);
    stop();
}

private void stop() {
    if (isMovingRight() == false && isMovingLeft() == false) {
        speedX = 0;
    }

    if (isMovingRight() == false && isMovingLeft() == true) {
        moveLeft();
    }

    if (isMovingRight() == true && isMovingLeft() == false) {
        moveRight();
    }

}
```

Kuva 12. Hahmon liikkeitä koskevaa koodia.

```
character = Assets.character;
character2 = Assets.character2;
character3 = Assets.character3;

anim = new Animation();
anim.addFrame(character, 1250);
anim.addFrame(character2, 50);
anim.addFrame(character3, 50);
anim.addFrame(character2, 50);

currentSprite = anim.getImage();
```

Kuva 13. Hahmon animaation määrittelyä GameScreen.java-luokassa

Tämän jälkeen piti hahmolle luoda peliin fysiikkasäännöt, jotta hahmo ei tippuisi läpi maan ja törmäisi seiniin eikä pystyisi ahtaissa tiloissa hyppimään. Lisäksi tippuminen piti ohjelmoida, jotta yksi pelin ehdoista toteutuisi (Jos hahmo tippuu aukosta, joutuu hän aloittamaan kentän uudestaan). Muutoin pelissä fysiikan sääntöjen kanssa on vähän olematonta, kuuluen pelin luonteeseen.

```
rect.set(centerX - 34, centerY - 63, centerX + 34, centerY);
rect2.set(rect.left, rect.top + 63, rect.left+68, rect.top + 128);
rect3.set(rect.left - 26, rect.top+32, rect.left, rect.top+52);
rect4.set(rect.left + 68, rect.top+32, rect.left+94, rect.top+52);
yellowRed.set(centerX - 110, centerY - 110, centerX + 70, centerY + 70);
footleft.set(centerX - 50, centerY + 20, centerX, centerY + 35);
footright.set(centerX, centerY + 20, centerX+50, centerY+35);

if (livesLeft == 0) {
    state = GameState.GameOver;
}
```

```
if (pixelson.getCenterY() > 500) {
    state = GameState.GameOver;
}
```

Kuva 14. Hahmoa koskevat fysiikkasäännöt

4.5 Peliäänien ja -musiikin luominen

Peliäänien ja -musiikin luominen tapahtui yhdistelemällä luokkia keskenään. Päädyin tekemään valikolle ja kentille musiikin sekä hahmolle äänet. Äänet ja musiikit eivät olleet välttämättömyys pelin kannalta, mutta päätin silti luoda ne jotta saisin tuntumaan minkälainen prosessi se tulisi olemaan peliprojektissa. Audio.java-luokassa luodaan äänentoistoa varten pohja, jotta äänien käyttöönotto onnistuisi myöhemmin.

```
package com.peksi.framework;

public interface Audio {
    public Music createMusic(String file);

    public Sound createSound(String file);
}
```

Kuva 15. Audio.java-luokka com.peksi.framework-paketissa, jolla luodaan äänentoisto peliä varten

```
package com.peksi.framework;

public interface Sound {
    public void play(float volume);

    public void dispose();
}
```

Kuva 16. Sound.java-luokan luominen

```
public interface Music {
    public void play();

    public void stop();

    public void pause();

    public void setLooping(boolean looping);

    public void setVolume(float volume);

    public boolean isPlaying();

    public boolean isStopped();

    public boolean isLooping();

    public void dispose();

    void seekBegin();
}
```

Kuva 17. Music.java-luokan määrittelyä. Public void dispose(); avulla voidaan poistaa musiikki tiedosto, joka vapauttaa puhelimen muistia käyttöön

Com.peksi.framework-pakettiin luoduilla luokilla saatiin äänelle ja musiikille luotua perustoiminnat, kuten play, stop ja pause. Tämä mahdollistaa äänien ja musiikin käyttämisen pelissä. Jotta nämä saataisiin käyttöön pelissä, tulee vielä luoda com.peksi.framework.implementation-pakettiin luokat, jotka ottavat nämä käyttöön.

Com.peksi.framework.implementation-pakettiin luodaan seuraavat luokat: AndroidAudio, AndroidMusic ja AndroidSound. AndroidAudio-luokassa otetaan käyttöön Audio-luokka implements-komennolla. @Override-komennolla estetään yliluokan metodien ylikirjoittumista. SoundPool-luokan avulla voidaan hallinnoida audion resurssien käyttöä joko muistin tai järjestelmän kautta. Lyhyet äänet voidaan täten ladata muistiin jonka avulla vähennetään CPU:n (prosessorin) rasitusta. Isoja äänitiedostoja (kuten taustamusiikki) käytetään järjestelmän kautta, sillä ne veisivät liian paljon muistia ja täten hidastaisivat puhelimen toimintaa.

```
package com.peksi.framework.implementation;

import java.io.IOException;

public class AndroidAudio implements Audio {
    AssetManager assets;
    SoundPool soundPool;

    public AndroidAudio(Activity activity) {
        activity.setVolumeControlStream(AudioManager.STREAM_MUSIC);
        this.assets = activity.getAssets();
        this.soundPool = new SoundPool(20, AudioManager.STREAM_MUSIC, 0);
    }

    @Override
    public Music createMusic(String filename) {
        try {
            AssetFileDescriptor assetDescriptor = assets.openFd(filename);
            return new AndroidMusic(assetDescriptor);
        } catch (IOException e) {
            throw new RuntimeException("Couldn't load music '" + filename + "'");
        }
    }

    @Override
    public Sound createSound(String filename) {
        try {
            AssetFileDescriptor assetDescriptor = assets.openFd(filename);
            int soundId = soundPool.load(assetDescriptor, 0);
            return new AndroidSound(soundPool, soundId);
        } catch (IOException e) {
            throw new RuntimeException("Couldn't load sound '" + filename + "'");
        }
    }
}
```

Kuva 18. AndroidAudio.java-luokka, jossa otetaan käyttöön äänet ja musiikit. SoundPool-luokalla hallinnoidaan audion (musiikki ja ääni) käyttöä joko puhelimen muistista tai suoraan tiedostosta

AndroidMusic.java-luokassa tuodaan käyttöön Music.java-luokasta useampi toiminta implements-komennolla. Luodaan myös olio MediaPlayer, joka hoitaa audion ja videon toistamisen Androidilla. MediaPlayerissä on monia Listener-toimintoja, joilla tarkistetaan toiston tila ja toimivat sen mukaisesti.

```
package com.peksi.framework.implementation;

import java.io.IOException;

public class AndroidMusic implements Music, OnCompletionListener, OnSeekCompleteListener, OnPreparedListener, OnVideoSizeChangedListener {
    MediaPlayer mediaPlayer;
    boolean isPrepared = false;

    public AndroidMusic(AssetFileDescriptor assetDescriptor) {
        mediaPlayer = new MediaPlayer();
        try {
            mediaPlayer.setDataSource(assetDescriptor.getFileDescriptor(),
                    assetDescriptor.getStartOffset(),
                    assetDescriptor.getLength());
            mediaPlayer.prepare();
            isPrepared = true;
            mediaPlayer.setOnCompletionListener(this);
            mediaPlayer.setOnSeekCompleteListener(this);
            mediaPlayer.setOnPreparedListener(this);
            mediaPlayer.setOnVideoSizeChangedListener(this);

        } catch (Exception e) {
            throw new RuntimeException("Couldn't load music");
        }
    }
```

Kuva 19. AndroidMusic.java-luokka, joka pitää sisällään musiikin toistoon liittyvät toiminnat

AndroidSound.java-luokassa tuodaan Sound.java-luokasta toiminnat käyttöön implement-komennolla. AndroidSound.java ei ole erityisen monimutkainen luokka; SoundPool:lla ja soundID:llä pidetään äänitiedostojen lukua yllä, toistetaan niitä ja poistetaan muistista dispose toiminnalla.

```
package com.peksi.framework.implementation;

import android.media.SoundPool;

import com.peksi.framework.Sound;

public class AndroidSound implements Sound {
    int soundId;
    SoundPool soundPool;

    public AndroidSound(SoundPool soundPool, int soundId) {
        this.soundId = soundId;
        this.soundPool = soundPool;
    }

    @Override
    public void play(float volume) {
        soundPool.play(soundId, volume, volume, 0, 0, 1);
    }

    @Override
    public void dispose() {
        soundPool.unload(soundId);
    }

}
```

Kuva 20. AndroidSound.java-luokassa tuodaan com.peksi.framework:n Sound.java-luokan toiminnot käyttöön

Äänille ja musiikeille on luotu nyt asetukset ja seuraavaksi ne otetaan käyttöön pelissä LoadingScreen.java- ja Assets.java-luokissa. Assets.java:ssa luodaan julkiset muuttujat fail.mp3- ja doing.mp3-tiedostoille, jotta muut luokat voivat niitä käyttää. Theme:lle määritetään sampleGame.getAudio().createMusic(”menutheme.mp3”)-toiminnolla musiikkitiedosto. Seuraavaksi toistetaanko sitä ja millä äänenvoimakkuudella. LoadingScreen.java:ssa alustetaan fail.mp3- ja doing.mp3-tiedostot peliä varten public void update toiminnalla. Täten pelin GameScreen.java-luokka voi käyttää niitä omiin tarkoituksiin.

```
public static Sound doing, fail;
public static Music theme;

public static void load(SampleGame sampleGame) {
    // TODO Auto-generated method stub
    theme = sampleGame.getAudio().createMusic("menutheme.mp3");
    theme.setLooping(true);
    theme.setVolume(0.85f);
    theme.play();
}
```

Kuva 21. Assets.java-luokassa luodaan muuttujat fail.mp3- ja doing.mp3-tiedostoille sekä määritellään menutheme.mp3-tiedoston asetuksia

```
Assets.doing = game.getAudio().createSound("doing.mp3");
Assets.fail = game.getAudio().createSound("fail.mp3");
```

Kuva 22. LoadingScreen.java:ssa alustetaan doing.mp3- ja fail.mp3-äänet pelin käyttöön

```
if (inBounds(event, 0, 285, 65, 65)) {
    pixelson.jump();
    Assets.doing.play(0.85f);
    currentSprite = anim.getImage();
}
```

Kuva 23. Doing.mp3-tiedoston toisto, kun pelaaja hyppää hahmolla

```
if (pixelson.getCenterY() > 500) {
    Assets.fail.play(0.85f);
    state = GameState.GameOver;
}
```

Kuva 24. Fail.mp3-tiedoston toisto, kun pelaaja ”kuolee”

4.6 Kentän luominen

Kentän luominen tapahtui com.peksi.sonofpixel paketissa jossa yhdistyivät Assets.java-, Tile.java- ja Background.java-luokat yhdeksi toimivaksi kokonaisuudeksi joka alustetaan LoadingScreen.java-luokassa SampleGame.java- ja GameScreen.java-luokkien käyttöön.

Aluksi luodaan kentälle .txt-tiedostolla rakenne, jonka päälle voidaan tehdä kentän ulkonäkö. Kuva 25:ssä näkyy kentän rakennetta, jossa jokainen rivi ja kolumni sisältää luvun, joka edustaa kentän ”maalaamiseen” käytettäviä tile.java-luokan olioita. Tyhjät kohdat edustavat tyhjää, joissa ei ole ole mitään hahmon liikkumista estäviä objekteja. Numeroilla 2, 4, 5, 6 ja 8 määritetään kenttää varten omat objektit, joilla luodaan ulkonäköä ja hahmolle alustaa, jotta se pystyisi liikkumaan kentällä. 2, 4, 5, 6 ja 8:n luvuille määritellään vielä erikseen .png-kuvatiedostot Tile.java-luokassa.

```
! Peksigaming
! SonOfPixel Level 1
! # = Pixel Legend
! 5 = HardPixel
! 8 = PixelGround
! 2 = PixelAir
! 4 = PixelLeft
! 6 = PixelRight
22222222222222222222222222222222222255555555555555555552222222222222222222222222222
                                    555555555555555555
                                    222222222222222222
                                       888888888888
                                    88855555555555588
                                    555555555555555888
                                 885555555555555555555
                                 555555555555555555555
888888888888888888888888    888888855555555555555555555888888888888888888    88
#         #         #         #         #         #         #         #         #
```

Kuva 25. Map1.txt-tiedosto, kentän tekninen rakenne

Tile.java-luokassa luodaan kentän toimivuutta varten vaadittavia asetuksia. Aluksi tuodaan import-komennolla asetukset Android-luokkakirjastosta ja Image-luokka com.peksi.framework-paketista. Kuva 26:ssa näkyy osa Tile.java-luokan määrityksistä, josta seuraavissa kuvissa lisää.

```
import android.graphics.Rect;

import com.peksi.framework.Image;

public class Tile {

    private int tileX, tileY, speedX;
    public int type;
    public Image tileImage;

    private Pixelson pixelson = GameScreen.getPixelson();
    private Background bg = GameScreen.getBg1();

    private Rect r;
```

Kuva 26. Kentän koodia määritellään Tile.java-luokassa

Seuraavaksi määritellään public Tile(int x, int y, int typeInt)-osiossa horisontaalisen (tileX) ja vertikaalisen (tileY) objektien kooksi 40 pixeliä. Tämä tapahtuu kertomalla tileX ja tileY:n 40:llä koska kentän rakenteessa käytetään aiempaa .txt-tiedostoa. Type:llä määritetään tileille eri tyypit (2,4,5,6 ja 8) joille ladataan if ja else if-ehdoilla tyyppiä vastaava kuva. Else:llä määritellään tyypiksi 0, jolle ei ole kuvatiedostoa, täten sitä ei ole olemassa ollenkaan. Tällä saadaan kenttään tyhjät tilat.

```
public Tile(int x, int y, int typeInt) {
    tileX = x * 40;
    tileY = y * 40;

    type = typeInt;

    r = new Rect();

    if (type == 5) {
        tileImage = Assets.hardpixel;
    } else if (type == 8) {
        tileImage = Assets.pixelground;
    } else if (type == 4) {
        tileImage = Assets.pixelleft;

    } else if (type == 6) {
        tileImage = Assets.pixelright;

    } else if (type == 2) {
        tileImage = Assets.pixelair;
    } else {
        type = 0;
    }

}
```

Kuva 27. Tile.java-luokan jatkoa. Määritellään Tile-oliolle horisontaalinen ja vertikaalinen koko sekä eri tile-tyyppejä vastaavat kuvat if- ja else if-ehdoilla

Update-metodia käytetään jokaisessa pelin loopissa ja siinä hoidetaan taustan nopeus (speedX = bg.getSpeedX() * 5;) ja horisontaalisten objektien nopeus. Lisäksi hahmon törmäysten tarkastus tapahtuu tässä metodissa. Background.java-luokassa määritellään lyhyesti taustan ominaisuudet kuten millä nopeudella tausta liikkuu verrattuna pelaajan

liikeeseen. Assets.java-luokassa ladataan resurseista type:n mukaiset kuvat, jotka alustetaan LoadingScreen.java-luokassa.

```
public void update() {
    speedX = bg.getSpeedX() * 5;
    tileX += speedX;
    r.set(tileX, tileY, tileX+40, tileY+40);


    if (Rect.intersects(r, Pixelson.yellowRed) && type != 0) {
        checkVerticalCollision(Pixelson.rect, Pixelson.rect2);
        checkSideCollision(Pixelson.rect3, Pixelson.rect4, Pixelson.footleft, Pixelson.footright);
    }

}
```

Kuva 28. Tile.java-luokan jatkoa. Update-toiminnalla määritetään taustan ja horisontaalisten objektien nopeus sekä hahmon törmäykset

Kenttä otetaan peliä varten käyttöön ensin SampleGame.java-luokassa, jossa .txt-tiedosto käännetään String:ksi public Screen getInitScreen()-metodissa, jossa kääntäminen tapahtuu InputStream is = getResources().openRawResource(R.raw.map1); map = converStreamToString(is); -komennoilla. Useamman kentän kääntäminen tapahtuu samalla tavalla. Tämän jälkeen kenttä otetaan käyttöön GameScreen.java-luokassa private void loadMap()-metodissa, jossa haetaan tilearray:sta tiedot.

5 POHDINTAA

Java-ohjelmointikielellä toteutettu mobiilipeli opinnäytetyönä oli yhdistelmä kunnianhimoa ja suuren riskin ottamista, sillä aiempaa kokemusta mobiilipelistä tai -ohjelmoinnista ei ollut enkä ollut Javan kanssa ollut koskaan muutoin tekemisissä kuin jonkun nettiselainpeliä pelatessa. Tavoite oli kuitenkin kunniahimoinen ja uskon että tämän opinnäytetyön tekeminen antoi näyttöä siitä, että osaan tehdä Javalla mobiilipeliä. Tästä on suurta etua ohjelmoinnissa, jossa Suomessa on peliala suuressa nosteessa ja peliammattilaisten kysyntä kasvaa koko ajan (Jantunen 2013, hakupäivä 18.10.2013).

Mobiilipeli jäi lopulta ulkonäöllisesti erittäin köykäiseksi, johtuen omien taitojen raijoitteesta tehdä hienoa grafiikkaa. Lisäksi asetetuista vaatimusmäärittelyistä jäi toteutumatta kentän vaihtumisen ja päättymisen ohjelmointi. Uskoisin, että mikäli joku toinen tai olisi ollut ryhmä tekemässä minun lisäksi tätä mobiilipeliä, olisi pelistä tullut hieno niin toiminnallisuuden kuin ulkonäön puolesta. Ryhmässä työskentely olisi lisäksi antanut mahdollisuuden minulle keskittyä pelkästään koodin kirjoittamiseen ja dokumentointiin, joka jäi erittäin vähäiseksi loppujen lopuksi. Yksintyöskentely toisaalta mahdollisti tietynlaisen riippumattomuuden, kun ei tarvinnut muita odotella että pääsisi jatkamaan työtään.

Javalla ohjelmointi ei liiemmin eronnut PHP ja HTML -ohjelmoinnista, sillä suurimmat erot tulivat lähinnä koodien saattamisesta yhdeksi toimivaksi kokonaisuudeksi ja kuinka ei tarvinnut tehdä useita eri tiedostoja. Javalla liitettiin eri komennoilla luokkia toisiin, joista muodostui toimivia kokonaisuuksia ja lopulta yksi suuri asia eli mobiilipeli. Tämä teki erityisesti ohjelmoinnin sujuvaksi ja nautinnolliseksi, lisäksi tähän kun yhdistettiin toimivat sovellukset (Eclipse, notepad++) niin ainut haaste oli kirjoittaa toimivat koodit.

Tästä opinnäytetyöstä en välttämättä usko olevan suuremmin apua tai esimerkkiä oikeaan mobiilipelinohjelmointiprojektia varten. Toki, jos on samassa lähtökohdassa kuin minä, tavoitteena tehdä kokeileva mobiilipeli eikä taustaa tai koulussa ole ollut erityisesti Java-kursseja, voi saada perspektiiviä omaan työhön ja mahdollisesti myös arvioida tätä kautta tulevan työn määrää. Tässä opinnäytetyössä toteutukseen

käytettyistä sovelluksista saattaa myös oppia tai jopa löytää ihan uuden sovelluksen, jolla saattaisi tehdä oman työn paremmin.

Opinnäytetyön prosessi oli haastava kaikin puolin. Motivaation ja omaan tekemiseen uskominen oli koetuksella, mutta pienistä asioista liikkeelle lähteminen ja sitä kautta onnistumisten myötä, pysyin lopullisessa aikataulussa mukana. Vaikka peli ei esteettisesti silmiä hivele, olen erittäin tyytyväinen oppimaani ja tekemiseeni. Kenties tulevaisuudessa pelien ohjelmoinnin ääressä, olen mukana oikeassa ryhmässä jossa vastuualueet ovat jaettu jokaiselle sopivaksi ja peliprojekti on erittäin mielenkiintoinen. On myös mahdollista, että vaikkei juuri pelialalle ura aukene, niin ohjelmoinnille on tarvetta sillä alan huipuista kilpaillaan tulevaisuudessa enemmän paitsi pelifirmoissa myös esimerkiksi hyvinvointialalla, kuten kuntoilu ja painonhallintabisneksessä (Hyttinen 2013, hakupäivä 10.11.2013). Odotankin tulevia ohjelmointiprojekteja innolla, olivat ne sitten peleihin liittyviä tai johonkin muuhun käytännöllisen sovellukseen liittyviä.

LÄHTEET

About Java ME 2013. Oracle. Hakupäivä 20.10.2013. <www.oracle.com/technetwork/java/javame/about-java-me-395899.html>
Android SDK 2013. Android Developers. Hakupäivä 1.9.2013. <www.developer.android.com/sdk/index.html>
Android, the world's most popular mobile platform 2013. Android Developers. Hakupäivä 1.11.2013. <www.developer.android.com/about/index.hmtl>
Audacity 2013. Audacity. Hakupäivä 1.9.2013. <www.audacity.sourceforge.net>
Cho, James 2013. Android Game Development Tutorial. Kilobolt. Hakupäivä 10.9.2013. <www.kilobolt.com/game_development_tutorial.html>
Eclipse 2013. The Eclipse Foundation. Hakupäivä 1.9.2013. <www.eclipse.org>
Hyttinen, Kati 2013. Peliammattilaisten kysyntä kasvaa – alan huipuista kilpaillaan jo hyvinvointibisneksessäkin. MTV. Hakupäivä 10.11.2013. <www.mtv.fi/uutiset/kotimaa/artikkeli/peliammattilaisten-kysynta-kasvaa---alan-huipuista-kilpaillaan-jo-hyvinvointibisneksessakin/2369858>
Jantunen, Heikki 2013. Pelialan kasvu on nyt Suomessa hyvin nopeaa. Verkkouutiset. Hakupäivä 18.10.2013. <www.verkkouutiset.fi/talous/peliala tekes neoagames-9914>
Lehmusvirta, Antti 2013. Peliala odottaa roimaa liikevaihdon kasvua. Kauppalehti. Hakupäivä 25.10.2013. <www.kauppalehti.fi/etusivu/peliala+odottaa+roimaa+liikevaihdon+kasvua/201310533033>
Leskinen, JP 2013. Ericsson: Älypuhelinten käyttö kasvaa huimaa vauhtia. Kauppalehti. Hakupäivä 12.11.2013. <www.kauppalehti.fi/etusivu/ericsson+alypuhelinten+kaytto+kasvaa+huimaa+vauhtia/201311562901>
Naalisvaara, Mikko 2013. Suomalaisyhtiön mobiilipeliä ladattu 100 000 000 kertaa. Yle. Hakupäivä 28.10.2013. <www.yle.fi/uutiset/suomalaisyhtion_mobiilipelia_ladattu_100_000_000_kertaa/6886322>
Notepad++ 2013. Don Ho. Hakupäivä 1.9.2013. <www.notepad-plus-plus.org>
Paint 2013. Microsoft. Hakupäivä 1.9.2013. <www.windows.microsoft.com/fi-fi/windows7/products/features/paint>
The Java® Language Specification 2013. Oracle. Hakupäivä 3.11.2013. <http://docs.oracle.com/javase/specs/jls/se7/html/index.html>